Spring 2014

Improving the Performance and Energy Efficiency for Mobile Cloud Computing

Seungbeom Ma
San Jose State University

Improving the Performance and Energy Efficiency for Mobile Cloud Computing

A Writing Project

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Seungbeom Ma

May 2014

Improving the Performance and Energy Efficiency for Mobile Cloud Computing

By

Seungbeom Ma

APPROVED FOR THE DEPARTMENT OF COMPUTER

SCIENCE SAN JOSÉ STATE UNIVERSITY

May 2014

Dr. Melody Moh	Department of Computer Science
Dr. Teng Moh	Department of Computer Science
Byoung An	Testing House, Inc.

ABSTRACT

Improving the Performance and Energy Efficiency for Mobile Cloud Computing

By Seungbeom Ma

Based on the worldwide high-speed networks and advanced hardware (e.g., multiple cores mobile processor, and various sensors), mobile software industries enthusiastically release advanced mobile applications. These phenomena cause mobile devices to break down the limitation of time and place. Mobile cloud computing provides the most convenient communication and effective working environment to humans. However, the fundamental hardware has technical difficulties to keep up advanced technologies and applications in mobile devices, which means that there is a gap between available hardware resource and the demand of complex applications in mobile devices. The limited hardware decreases the quality of service. Mobile Cloud computing with computation offloading algorithms can alleviate current concern in mobile device industries. This paper proposes a Dynamic Threshold Algorithm (DTA), which is an formulated algorithm to offload tasks in workflow to either the cloud environment or a local mobile device. Experimental results will prove that DTA is able to maximize the performance and minimize the energy consumption for mobile devices.

ACKNOWLEDGEMENTS

I would never have been able to finish my thesis project without academic supporting and wise advice from my advisor, committee members and support from my family and friends and teammates.

I would like to deeply appreciate to my advisor, Dr. Melody Moh for her teaching, encouragement, and kindness for the completion of this project. I was able have very special experience to enhance my knowledge of Computer Science area.

I would also like to thanks the San Jose State Department of Computer Science, Dr. Teng Moh, Byoung An for all their support to provide me a knowledge and opportunity to gain unique experience in the field of Computer Science.

Last, I deeply appreciate my parent's, my sister's and classmates' supporting and encouragement. I would never achieve this thesis project without their supporting. Specially, I would like to thank my parents. They were always willing to help and give me the best suggestion for my study and life.

TABLE OF CONTENTS

CHAPTER PAGE

1. **INTRODUCTION…….....11**

2. **RELATED WORK…….....14**

3. **DEFINITIONS AND ALGORITHM…….....16**

   3.1. **TASKS VISUALIZATION: WORKFLOW……....16**

   3.2. **DYNAMIC THRESHOLD ALGORITHM…….16**

4. **EXPERIMENT SET UP……...20**

   4.1. **EXPERIMENT PLATFORM ARCHITECTURE…….20**

   4.2. **HARDWARE AND TEST ENVIRONMENT SETTING……..22**

   4.3. **TIMES AND ENERGY TRAKER SETTING……24**

   4.4. **APPLICATION: CUSTOMIZED GAUSSIAN METHOD…….. 26**

   4.5. **SOFTWARE SETTING……27**

5. **EXPERIMENTAL RESULTS……...…..29**

6. **CONCLUSION AND FUTURE WORK……...…..34**

LIST OF REFERENCES……...…..35

LIST OF ACRONYMS

CBIR	Content-Based Image Retrieval
DAG	Directed Acyclic Graph
DTA	Dynamic Threshold Algorithm
EMSO	Energy-Efficient Multisite Offloading algorithm
GAE	Google App Engine
IaaS	Infrastructure as a Service
JVM	Java Virtual Machine
MCC	Mobile Cloud Computing
MQMW	Multiple QoS constrained scheduling strategy of Multi-Workflows
PSO	Particle Swarm Optimization
PaaS	Platform as a Service
STA	Static Threshold Algorithm
SaaS	Software as a Service

LIST OF FIGURES

FIGURE **Page**

Figure 1: Abstraction of Mobile Computing……..12

Figure 2: Workflow……16

Figure 3: Estimator Function Pseudocode…….19

Figure 4: Abstractions of Cloud Computing Services……...20

Figure 5: Main Interface of Project Mobile Cloud Application……21

Figure 6: Smartphone Setting……23

Figure 7: Test Environment Setting……..24

Figure 8: Watt Electricity Usage Monitor with Smartphone……25

Figure 9: Power Tutor Interface……25

Figure 10: Customized Gaussian Function Pseudocode……...26

Figure 11: Sample Experimental Workflow 50……27

Figure 12: Tasks Distribution……...……28

Figure 13: Test Result for Balanced and Extreme Workflow 25……..30

Figure 14: Test Result for Balanced and Extreme Workflow 50……..31

Figure 15: STA vs DTA Workflow 25……32

Figure 16: STA vs DTA Workflow 50…….33

LIST OF TABLES

Table **Page**

Table 1: Average Execution time and Time Weight for 1 Cycle in Cloud and Local Mode…….18

1. INTRODUCTION

Based on 3G and 4G networks, the mobile devices become powerful tools to access Internet and advanced mobile applications. Smartphones, which belong to one significant category in mobile devices, were invented approximately twenty years ago to replace traditional mobile phones. These devices get other additional functions such as high-quality camera mobile, mobile apps that aid productivity, digital content streaming, web browsing, augmented reality and etc. They function like mini-computers, and they are tiny enough to fit in people's pockets. They have become an important part of our lives. Smartphone users expect their smartphones to run sophisticated and powerful applications. However, these functions regularly require complex mathematic calculation, which is one of the biggest factors to drain smartphones' battery. There are several power-conservation techniques [1,2] that temporarily clock down the CPU and change screen brightness during idle time. The smartphone vendors urgently solve these technical issues and make an improvement. Two temporary solutions accompany three primary trade-offs: cost, size and performance; moreover, they will not be critical solutions to satisfy mobile industries seeking for future smartphone trends. First, down clocking CPU speed becomes a major issue to run high performance CPU required to run mobile applications. The customers might be able to use the newest features and applications for a limited time. Second, smartphone users lose portability and capability. Third, smartphone venders sacrifice smartphones' designs in order to adopt bigger size of batteries.

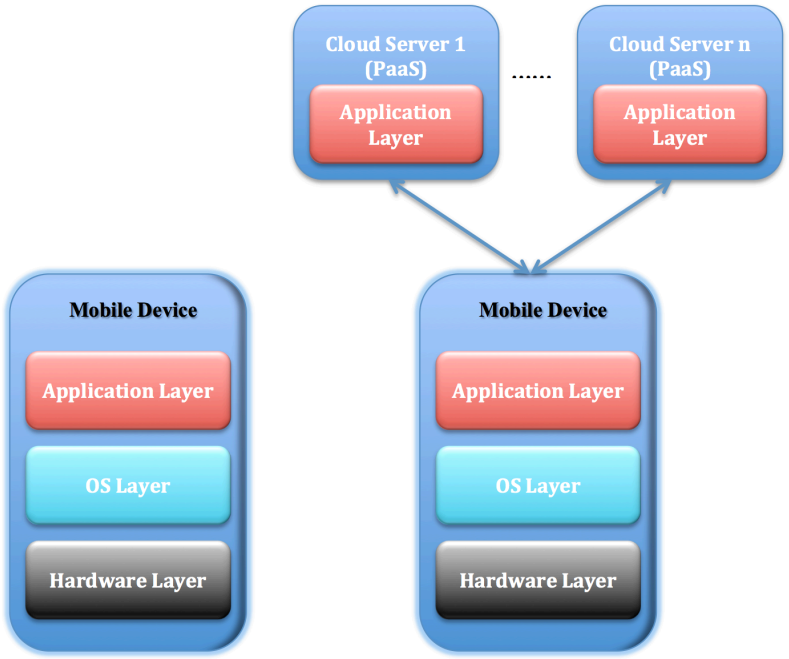


Figure 1: Abstraction of Mobile Computing
(a) Multiple mobile applications run on a mobile device without cloud computing (b) Mobile cloud by computation offloading from a mobile devices to N number of cloud servers

However, the mobile cloud computing which adopts concept of offloading can be a solution to break the limitation of mobile devices' capability. In our research, we adopt these concepts and implements DTA. Figure1 (a) describes that mobile devices traditionally execute tasks by itself due to limitation of network speed and coverage. However, Figure1 (b), mobile devices dynamically transmit large amount of computational tasks without any restrictions. DTA prioritizes tasks and make a decision whether mobile devices require offloading a task to cloud servers or not. Therefore, this paper will describe four major areas to prove that mobile cloud computing will be a future of mobile computing. First, we introduce an experiment platform and architecture for mobile and cloud experiment. Second, we will introduce the concept of testing

application such as tasks and a workflow. Third, this project introduces a mobile application and implements Dynamic Threshold Algorithm (DTA). It adopts the concept of offloading algorithm to remote multiple cloud servers and distributing tasks to a local device and cloud servers; moreover, this project implements another algorithm to prove that DTA can be a possible solution to keep powerful performance of mobile devices. Finally, we will analyze the experiment results to prove that DTA can be a possible solution for mobile computing.

2. RELATED WORK

The research area of offloading algorithms consists of three main subjects, which are the areas of making offloading decisions, applying the first subject in reality, and building offloading infrastructures [3]. This section discusses the related work for the first subject of offloading algorithms research area.

Offloading algorithms are different from the traditional mobile computing, which always run computation in mobile devices or transmit computation to a single server. The principle of offloading algorithm adopts the opposite concept of grid computing. For example, FightAIDS@home [4], SETI@home [5], and Floding@home [6] distribute tasks to several users to get computational assistances. These services apply the resources of several thousand of users in a network. On the other hand, offloading algorithms apply several users, which are in a network, to request computational performance from several cloud computing servers. These algorithms make a decision based on costs and makespan, which are the size of computation (e.g., mathematic calculation, and image retrieval), the size of data packet (e.g., sequence of string and size of image) and the network bandwidth [7][8][9]. Hoang Chonho, Dusit, and Ping researched and defined Mobile Cloud Computing (MCC)[10]. They basically discussed the overview of the architecture and the application. Hong, Kumar, and Lu studied offloading computation algorithm for mobile devices, which assist content-based image retrieval (CBIR)[7]. Nimmagadda, Kummar, and Lu [8] researched and experimented in offloading system to track and recognize moving an object in real-time. Wolski, Gurun, and Krintz [9] studied and experimented with the framework, which made computation

offloading decisions on local cost and remote cost.

Several offloading and cloud based algorithms adopted workflow to visualize tasks in mobile applications. Niu, Song, and Liu studied an Energy-Efficient Multisite Offloading Algorithm (EMSO)[11] which utilized workflow to visualize the amount of tasks, tasks' size, and dependency between tasks in a mobile application. Xu, Cui, Wang, and Bi also used a sample strategy to implement and test Multiple QoS constrained Scheduling Strategy of Multi-Workflows (MQMW)[12] to compute more than one workflow at the same time.

In conclusion, the algorithms and the concept of workflow mentioned above maximize the performance of a mobile device and a cloud server by minimizing the makespan and several types of costs. It motivated me to design DTA. However, we would apply this algorithm to the physical world with familiar environment and devices. To adopt computation offload encountered many challenges. First, DTA needed to implement or search proper applications and cloud services, which possibly required reasonable execution time and power in a mobile device and a cloud server. Second, we needed to figure out the way to split the tasks in an application. Third, tracking execution time and power consumption was one of the biggest challenges in this project. These three challenges were crucial parts to implement DTA because it would affect the offloading scenario to execute the tasks in a mobile application or a cloud server. These three subjects will be described in the next part.

3. DEFINITIONS AND ALGORITHM

3.1 TASKS VISUALIZATION: WORKFLOW

To visualize tasks' relations and determine the portion of computation for offloading, we design workflow graphs to input data for a mobile application. This workflow utilizes the concept of Directed Acyclic Graph (DAG). The workflow consists of more than one nodes and a single root node. Each node has dependence, so each node is connected by edge. A node in Figure 2 consists of a key-value pair such as "Task_ID" and "Task_Repetition". First, "Task ID" is a unique identifier for each task. Once an application holds a task, DTA will offload the task to the cloud or execute it in a mobile device. After a mobile device accumulates results and puts it together, "Task_ID" purposes to prove and display the results. Second, "Task_Repetition" will represent the size of a node. This value will be the key criterion to decide whether the current task is suitable to run on smartphones or not.

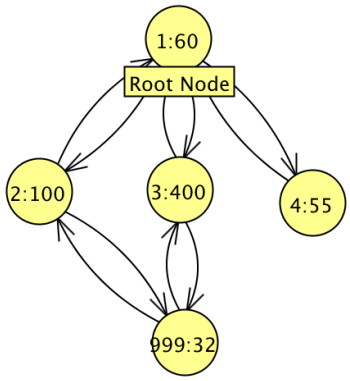


Figure 2: Workflow

3.2 DYNAMIC THRESHOLD ALGORITHM

DTA is based on the physical measurement to handle the corner cases of STA. In

reality, mobile applications possibly produce both large amount of small computational tasks and larger computational tasks in real-time. Therefore, it is too expensive to run all small tasks in a mobile device, whereas a mobile device possibly has long idle time to run all the tasks in cloud servers. This algorithm predicts total execution time in order to make a computational balance between a mobile device and cloud servers, which means that it equally distributes computation to get rid of idle time in a mobile device and cloud servers. DTA travels around the workflow based on Breadth-first search algorithm to collect the nodes. While DTA is collecting nodes, DTA keeps tracking the estimated total execution time for a mobile device and each cloud server. Based on the total estimated execution time, DTA will decide places to assign a task.

DTA is based on two core functions to maintain the performance of a mobile device and cloud severs. First, "get_weight" function in Figure 3 returns time weight based on our collection of raw data. We purely run a single task with "Local" and "Cloud" mode and uses Equation 1 to calculate the time cost of a single cycle to determine execution time. In Table 1, we can observe single repetition time cost difference between purely running a single task in a local device and doing so in a cloud server. This difference generates the ratio of time weight to estimate the time cost of a task. The "n" in Equation 1 and 2 designate a target device. The "$T(v)_0$" indicates the time cost of "Local Mode" and the "$T(v)_1$" indicates the time cost of "Cloud Mode" in a single task.

$$[\text{Average Time for Single Repetition}]_n = T(v)_n \,/\, \text{Size}(v)\,, v \in V$$

Equation 1: Average Time Consumption per cycle

$$Sum(Time_{est})_n = T(D_n)_{tot} * (\gamma_n)_m * Equation1 * Size(v), v \in W$$

$$(\gamma_n)_m = \begin{cases} (\gamma_n)_m = 1 \text{ if } n < 1 \\ \text{otherwise } (\gamma_n)_m \end{cases} \text{ [n: Target Device, m: Task Repetition]}$$

Equation 2: Total Time Cost Estimation

In Equation 2, D_n indicates the estimated total time cost of the target device "n". "n =0" will target a mobile device and other numbers of "n" will target cloud servers. To equalize the estimate time, local device execution will be a standard to compare time cost between a mobile device and cloud servers. Weight will be only applied for estimating execution time for "Cloud Mode". Based on Table 1 and Equation 2, DTA estimates total execution time for a mobile device and each cloud server. The estimation is the sum of the previous total time cost estimation and the current node's time cost estimation.

Table 1: Average Execution time and Time Weight for 1 Cycle in Cloud and Local Mode

Task Repetition	Cloud Mode (s)	Local Mode (s)	Time Weight
50	0.005287	0.00147	3.596598639
100	0.002521	0.00194	1.299484536
150	0.001697	0.001355	1.252398524
200	0.001289	0.001371	0.940189643
250	0.000982	0.001362	0.720998532
500	0.000655	0.001307	0.501147666
1000	0.000511	0.001348	0.379080119
1500	0.000437	0.001316	0.332066869
2000	0.000405	0.001305	0.310344828
3000	0.000378	0.001308	0.288990826
4000	0.000359	0.001319	0.272175891

```
1. Function Time_Estimator:
2.     Input: Local_est , Cloud1_est , Cloud2_est, task
3.     Output: 0-> Execute in smartphone
4.             1 -> Execute in Cloud server 1
```

```
5.              2 -> Execute in Cloud server 2
6.      weight = get_weight(task)
7.      Est_tot[0] = Local_est + task
8.      Est_tot [1]  = Cloud1_est + (weight*task)
9.      Est_tot [2]  = Cloud2_est + (weight*task)
10.     Min_est = Min(Est_tot)
11.     IF Min_est == Est_tot[0] THEN
12.         Local_est = Min_est
13.         RETURN 0
14.     ELSEIF Min_est == Est_tot[1] THEN
15.         Cloud1_est = Min_est
16.         RETURN 1
17.     ELSE
18.         Cloud2_est = Min_est
19.         RETURN 2
20.     ENDIF
21. END FUNCTION
```

Figure 3: Estimator Function Pseudocode

From Psedocode in Figure 3, Time_Estimator function keeps tracking the total amount of execution time to balance between each cloud server and a smartphone. This algorithm is working by following steps mentioned below. First, in line 6 to 9, Time_Estimator calls "get_weight" function to compute the total time length of a smartphone and cloud servers. These time costs will be assigned into Est_{tot} list. Second, in the line 11, Time_Estimator searches for the smallest value in the Est_{tot} and assigns the smallest value in Min_{est}. Third, Time_Estimator compares the values in Est_{tot} list and Min_{est} to search a device to assign a task. Finally, Time_Estimator goes back to line 1 to consume all the tasks in the workflow.

4. EXPERIMENT SET UP

4.1 EXPERIMENT PLATFORM ARCHITECTURE

This project aims to bring the theoretical algorithm to real cloud and mobile environment. This project selects Google App Engine (GAE) and an Android OS based smartphone to measure execution time and energy consumption. Using a real device and cloud computing environment can be a good chance to observe the actual performance of mobile cloud computing algorithm. We are able to get real execution time and energy consumption through a smartphone.

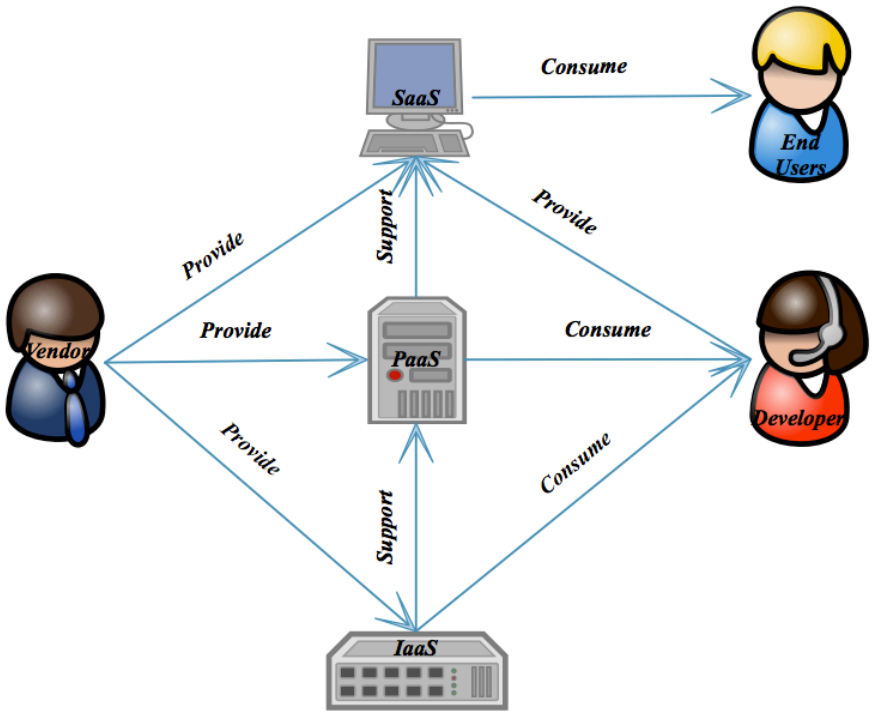


Figure 4: Abstractions of Cloud Computing Services

For the cloud server side, this project selects the GAE, which is Platform as a Service (PaaS) in Google. PaaS is an outgrowth of Software as a Service, a software

distribution model in which hosted software applications are models available to customers over the Internet. [7][13] In abstractions of cloud computing services in Figure 4, the vendors provide a full or partial application development system. The applications will be ran and stored in the service provider data centers, so developers do not have to concern with platform and data storage.

GAE is one of most popular PaaS to develop and host a web application in the cloud market. GAE aims to provide the web application hosting service, which is able to develop and deploy a web application within a pre-defined runtime environment. Windows Mobile OS, and IOS requires the understanding of how to cross the Java based platform and apply it to different language platform. To build Android application, we can obtain full advantages from Google developing environment. Based on Google's PaaS with JVM, we implements and deploys a small Java based application for Android and a cloud server in parallel.

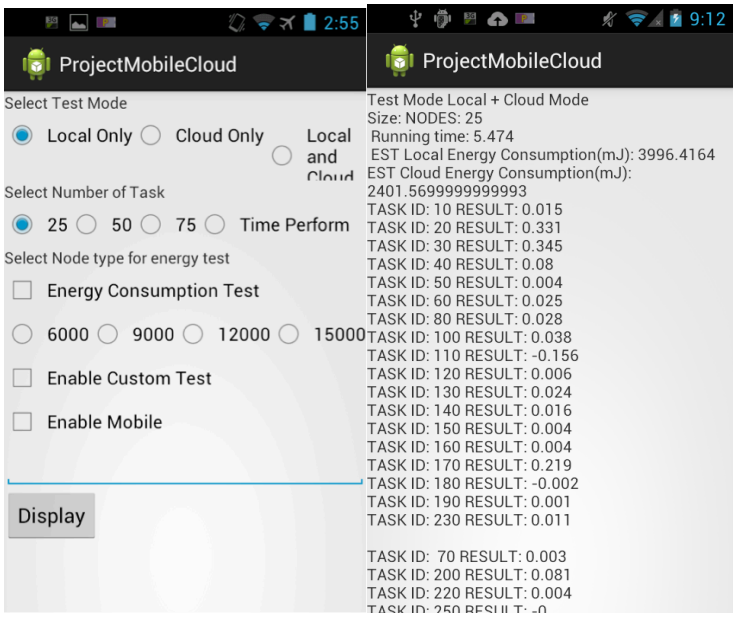


Figure 5: Main Interface of Project Mobile Cloud Application
Front Interface (Left) and Result Display Interface (Right)

Figure 5(Left) represents the basic interface of an application for the smartphone side. This application is able to run three types of testing modes. First, “Local Only” mode runs tasks in workflow on the local mobile device. Second, “Cloud Only” mode offloads all the tasks to the cloud-computing environment, which means that a mobile device transmits entire tasks in workflow to a cloud computing server, and an application in a cloud computing server return results to a mobile device. Third, “Local and Cloud” mode executes tasks for Static Threshold Algorithm (STA) or Dynamic Threshold Algorithm (DTA). “Select Node type for energy test” will not be used in our experiment. This project uses this section to test basic functionalities for both a cloud and a mobile application and to get raw data for a single task. The right side of Figure 5 displays the result of each test mode. It displays the basic information of the test and the results to prove that a mobile device has communication with GAE.

4.2 HARDWARE AND TEST ENVIRONMENT SETTING

This section discusses the hardware and the environment setting for testing. For the cloud server side, we used two F1 instance classes in GAE. This instance consisted of 600Mz CPU and 128 MB ram. Max idle instances and Min pending latency were set in an automatic mode. For the client side, we used a Google reference smartphone and a Motorola router for testing. In Figure 6, we used a Nexus S (Unlocked) with 4.0.4 version of Android OS. For the Wi-Fi setting, we used a Motorola router, which supports Wi-Fi 802.11g. To build a constant test environment, we tested DTA, and STA at 1:00 am to minimize the network traffic.

Figure 6: Smartphone Setting

As you can see in Figure 7, we set 0.5 feet distance between a mobile phone and Wi-Fi router. Battery was fully changed and a smartphone was connected with a batter charger. We killed and uninstalled unnecessary applications in the Nexus S to prevent potential system resource leaking and restarted the smartphone after 20 times of testing.

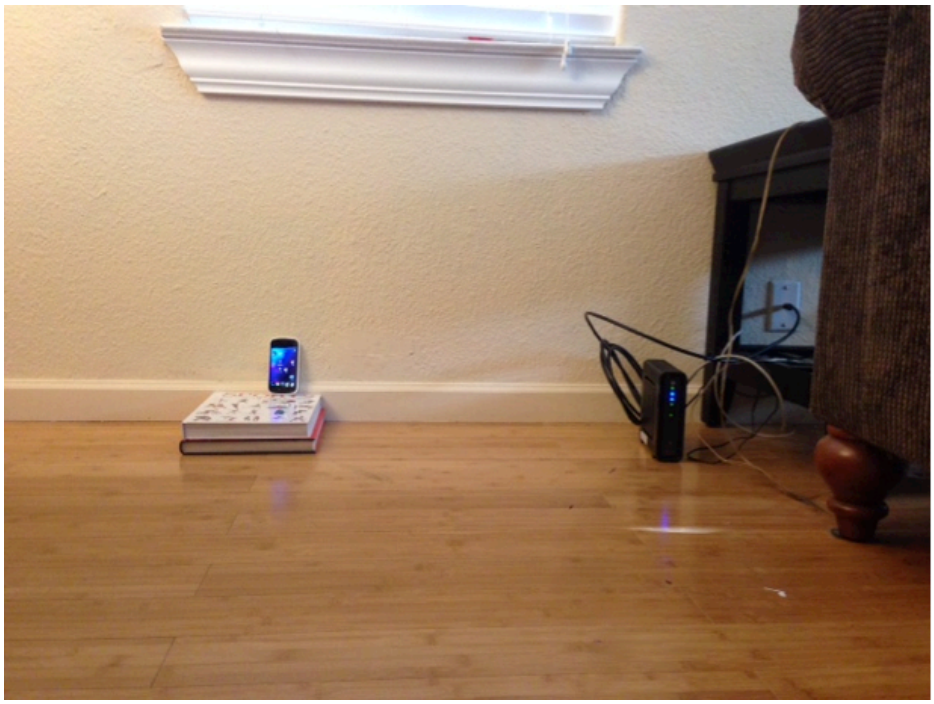


Figure 7: Test Environment Setting

4.3 TIMES AND ENERGY TRAKER SETTING

To track the execution and the energy consumption, we used both Java API and energy consumption application. First, we used system time in Android OS to determine the execution time. Second, we used both virtual and physical methods to measure power consumption. In the physical method, we used "A Watt Electricity Usage Monitor" to measure power consumption in Figure 8.

However, there were minor technical issues to measure power consumption. This device was unable to monitor the detailed power consumption of a smartphone, but we were still able to observe that mobile device consumes power. In the virtual method, we used "Power Tutor" application to measure the power consumption while we were running a test application. Power Tutor was developed by University of Michigan Ph.D

students. This application was for analyzing power consumed by major system components in mobile devices, which included a CPU, a network interface, a display, and a different application. As we can see in Figure 9, we used the “Pie view” mode to track the power consumption of CPU and wireless technology.

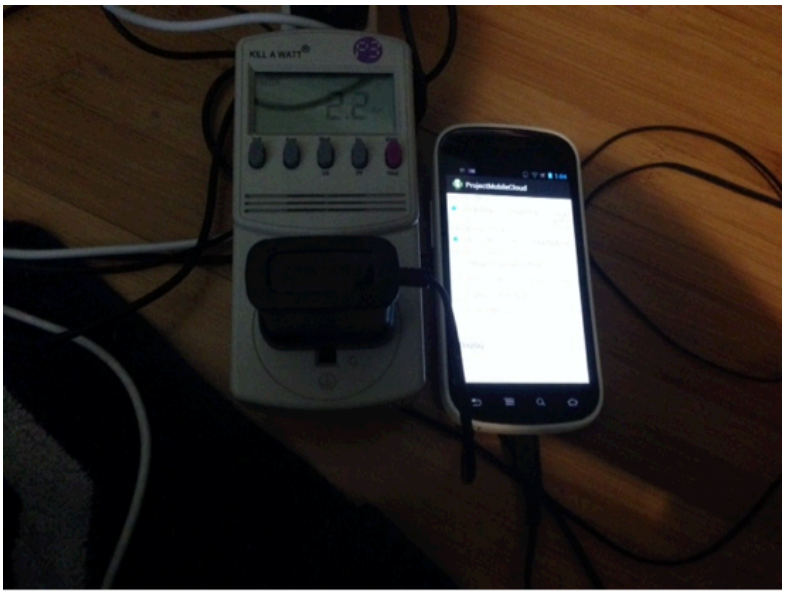


Figure 8: Watt Electricity Usage Monitor with Smartphone

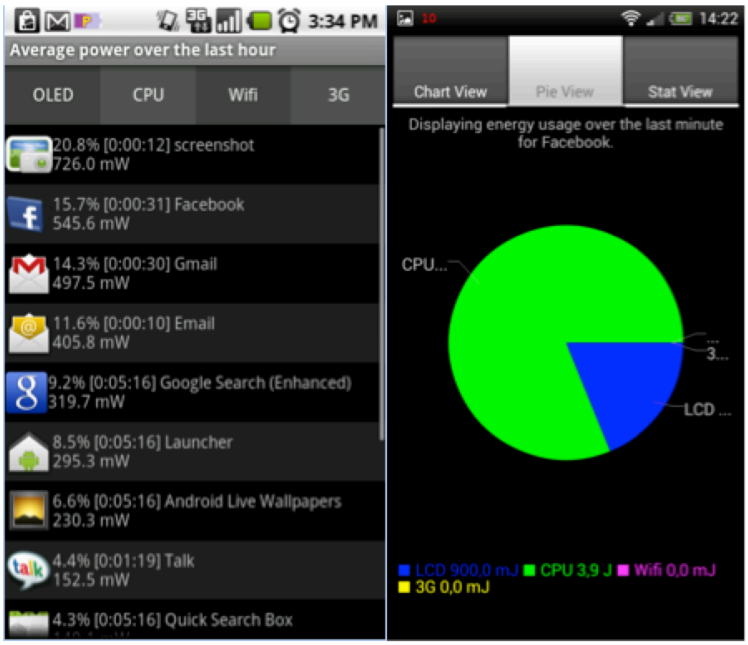


Figure 9: Power Tutor Interfaces
Task Manger Interface (Left), Pie View Interface (Right)

4.4 APPLICATION: CUSTOMIZED GAUSSIAN METHOD

For consuming energy and time on a local device and a cloud server, we applied Gaussian application to solve mathematic computations. We used this application to compute the Gaussian probability density function and the Gaussian cumulative density function [14]. However, Gaussian function did not produce enough time and energy usage for experiment. Therefore, we added an additional mathematic tangent function for mobile device burdens.

```
1.  Function Gaussian_AVG:
2.          Input: Process_List <Task_ID: Task_Repetition >
3.          Output: Gaussian Average Results
4.     For i = 0 to Process_List.length-1
5.           For j = 0  to   Task_ Repetition
6.       //+1 to prevent to pass 0 value in parameters
7.                  zeta = RadomDouble.next() + 1
8.                  mu =  RandomDouble.next() + 1
9.                  sigma = RandomDouble.next() + 1
10.                 Sum += Gaussian(zeta, mu, sigma)
11.          Next
12.          Result_list[i]=(Task_ID, Sum/Task_ Repetition)
13.   Next
14. Return Result_list
```

Figure 10: Customized Gaussian Function Pseudocode

The function in Figure 10, “Task Repetition” will be the amount of work to run Gaussian Function for each task. For example, when the function gets “Process_List = 1” and “Task_ Repetition = 100”, then Gaussian function gets one task and will loops from 0 to 100 to calculate the average value of Gaussian computation. We assume that each task in the application should have unique execution time and power consumption because each smartphone user has a different pattern to use a mobile application.

Therefore, we randomized three parameters, which are “zeta”, “mu”, and “sigma”. This function will not only return unique results but also generate unique power consumption and execution time.

4.5 SOFTWARE SETTING

This section presents the experimental setting for client side. We tested three major algorithms STA, DTA, and Local Mode. We ran balanced and extreme 25, and 50 tasks of workflows. The total amount of repetition would be 5380 times for workflow 25 and 14500 times for workflow 50. These algorithms would the dataset used in Figure 11 and 12.

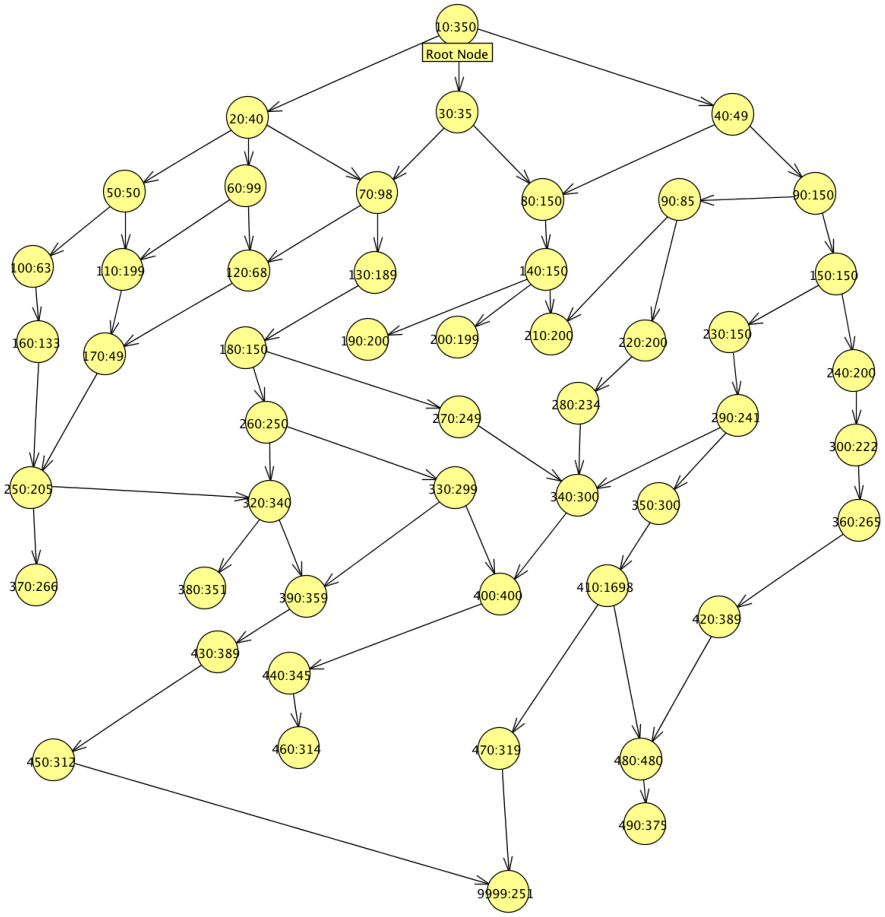


Figure 11: Sample Experimental Workflow 50

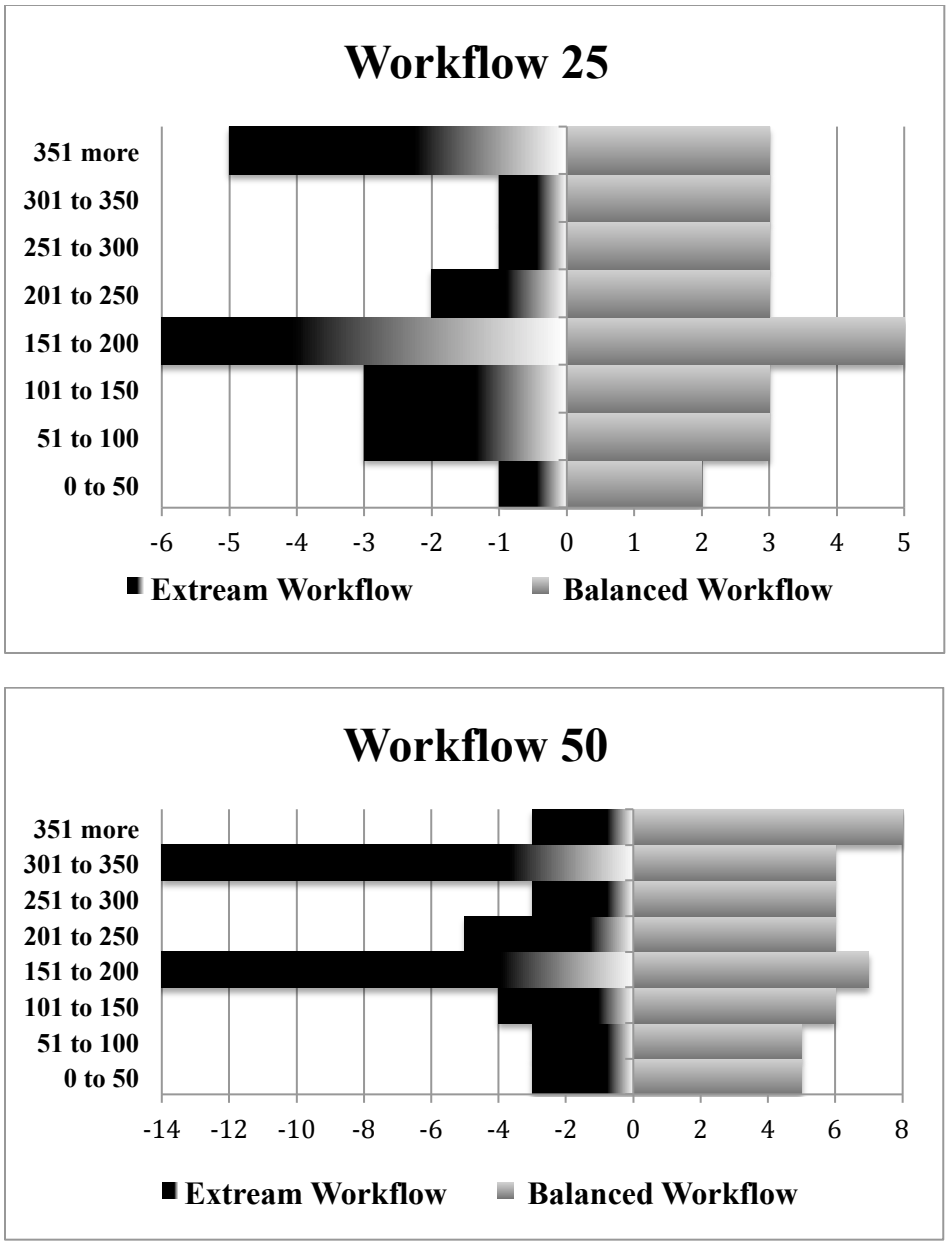


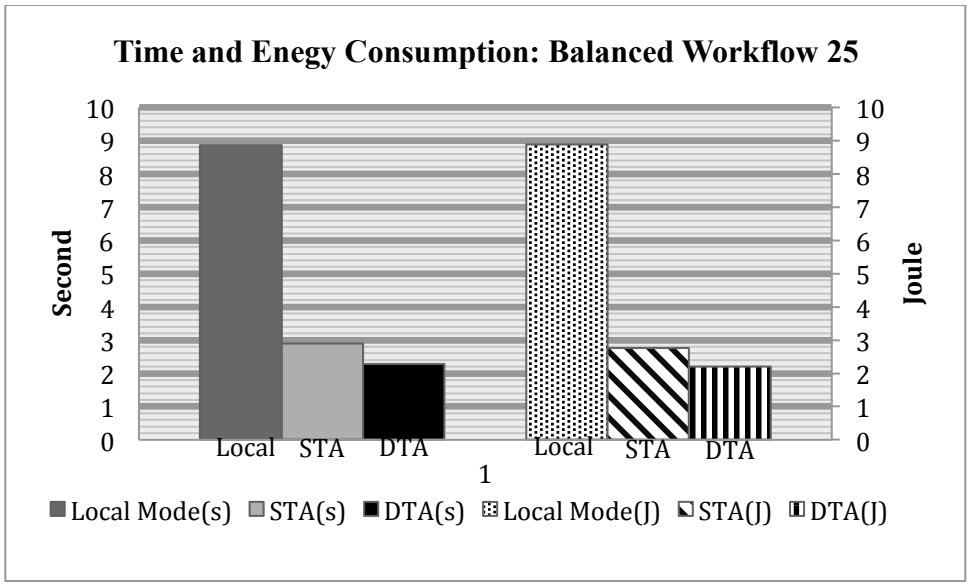
Figure 12: Tasks Distribution

Size 25(Above), and Size 50(Below)

5. EXPERIMENTAL RESULTS

This section discusses the experimental results of DTA. From the overall results in Figure 13, we can determine that STA and DTA spend approximately 300 percent less energy than Local Mode. In Figure 14, the workflow 50 shows that Local Mode consumes approximately 500 percent more than STA and DTA. These two results suggest that STA and DTA consistently execute the large amount of tasks in workflow, whereas Local Mode rapidly drains CPU resource and battery. It proves that mobile cloud algorithms effectively distribute tasks to a mobile device and cloud-computing servers.

DTA spends less time and energy for small and large workflow. As shown in Figure 15, DTA spends approximately 24 percent less time than STA; moreover it improves approximately 8 percent of energy efficiency on the balanced workflow. However, in the extreme case, DTA saves 21 percent. The results on Figure 16 indicate that STA begins increasing time and energy consumption while it runs the extreme workflow. We can observe that STA consumes 28 percent more time and power for executing tasks in the extreme workflow. However, DTA keeps constant time and energy consumption, so we do not observe significant usage increment between the results of balanced workflow and extreme workflow. This is because DTA successfully find the shortest execution time among the mobile device and cloud servers.



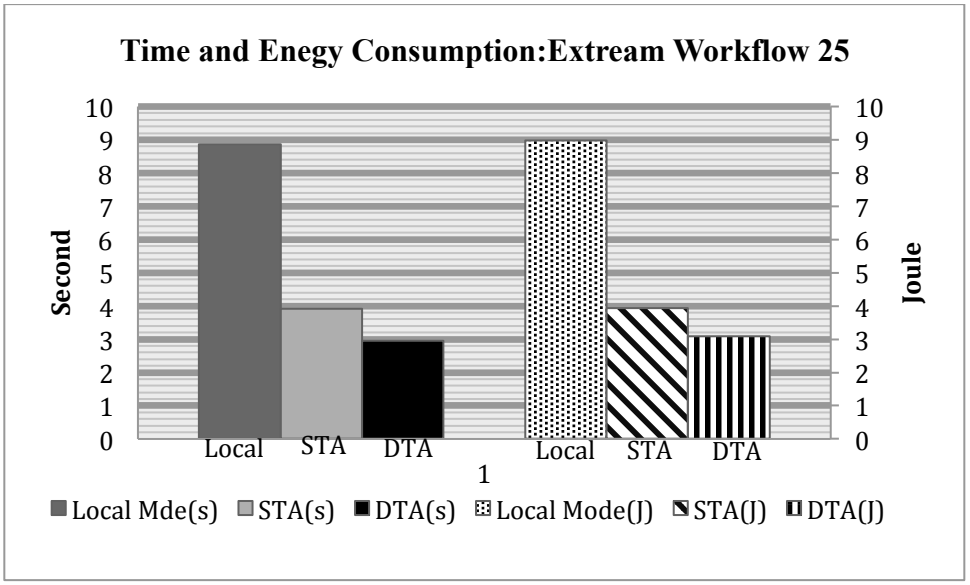
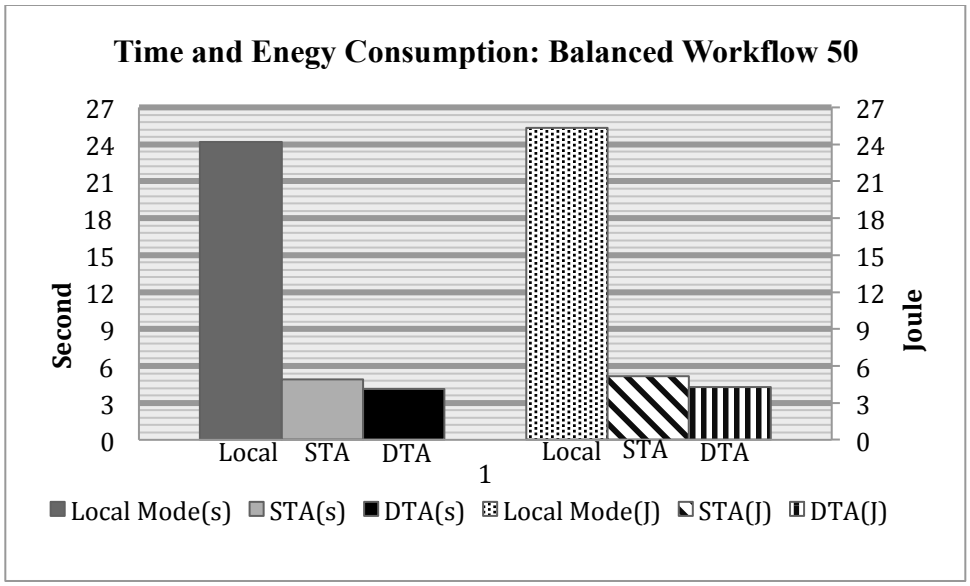


Figure 13: Test Result for Balanced and Extream Workflow 25



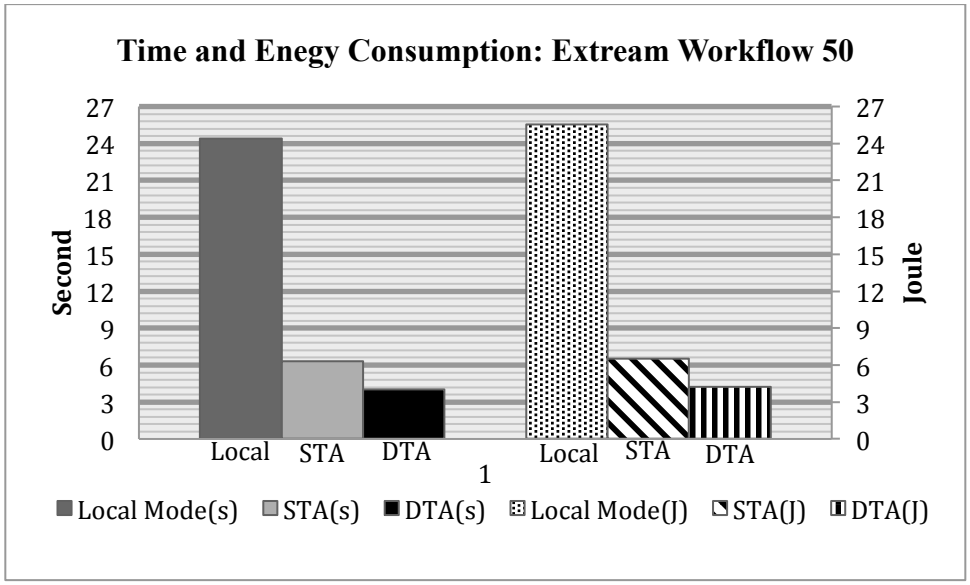
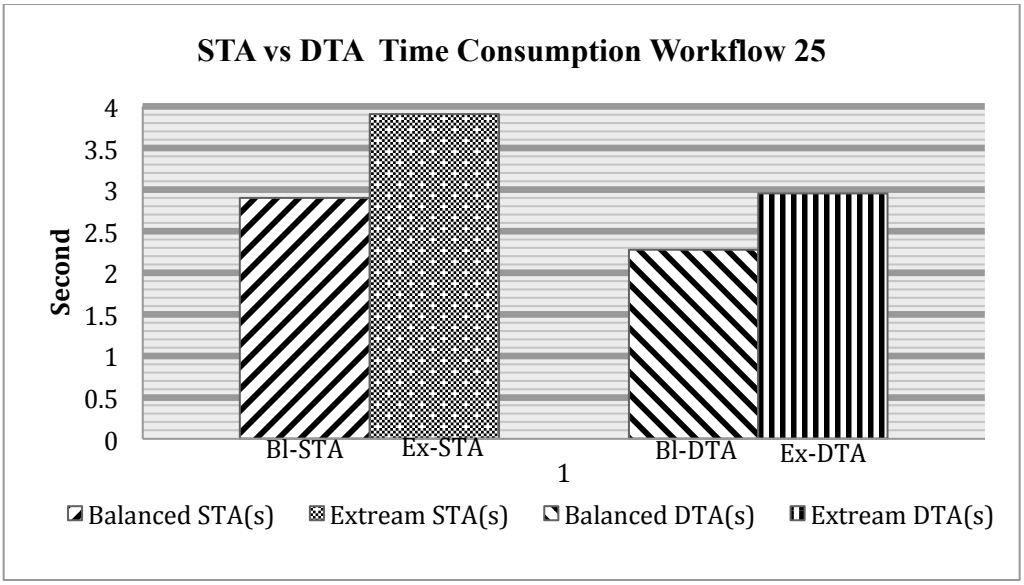


Figure 14: Test Result for Balanced and Extreme Workflow 50



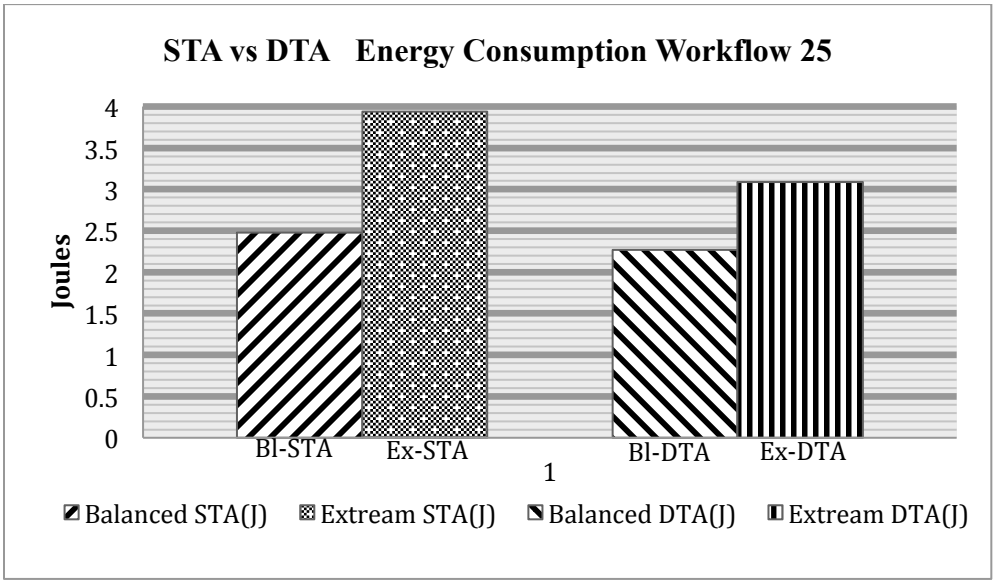
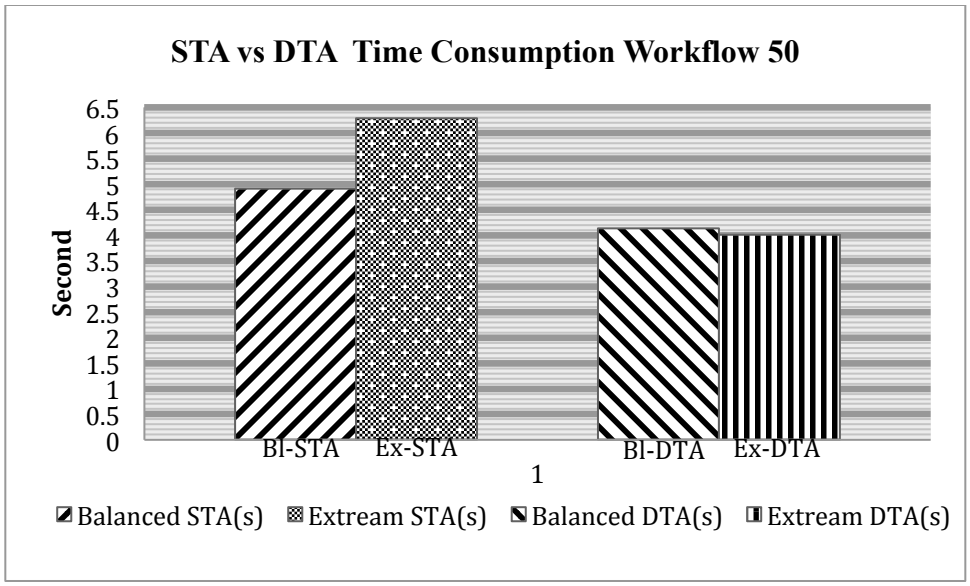


Figure 15: STA vs DTA Workflow 25



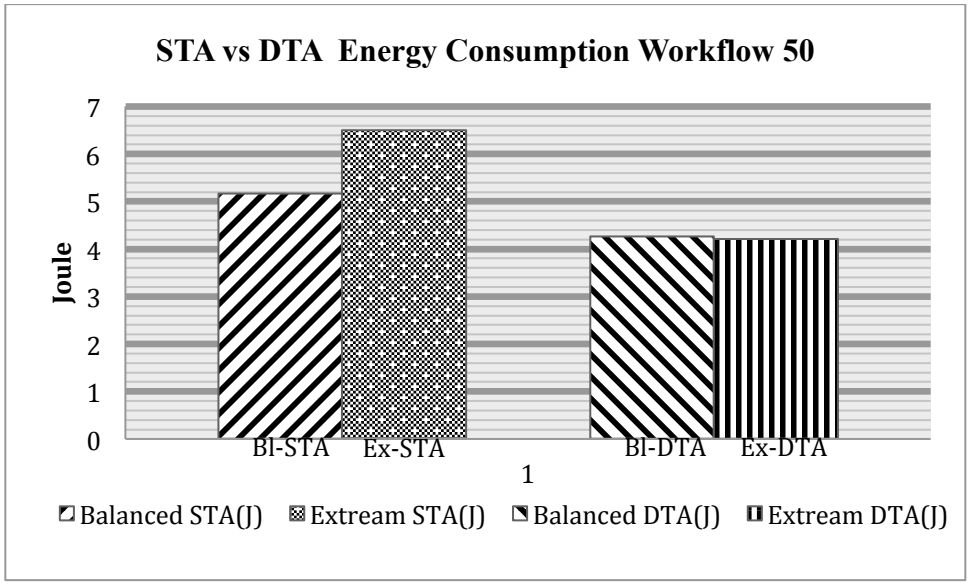


Figure 16: STA vs DTA Workflow 50

6. CONCLUSTION AND FUTURE WORK

This paper intends that DTA for mobile cloud computing maximize time and energy efficiency. The experimental results mentioned above demonstrate that DTA obviously saves the most energy and maximizes performance of mobile device. It minimizes idle time in cloud servers and mobile devices to maximize performance and conserve energy consumption. With the idea of DTA and workflow, we will improve the current algorithm based on three topics as a future work. First, DTA will adopt searching algorithms to maximize the performance of DTA. The tasks in the workflow have the dependency relationship, so we can add the network condition to define the relationship between tasks to implement searching algorithms, such as PSO (Particle Swarm Optimization) and MQMW [12][15]. Second, DTA adopts multiple computational applications in cloud servers and mobile applications. Handling several types of computational applications will solidly optimize DTA and prove that DTA will be a possible solution for future of mobile cloud computing. Third, we will build a function to make data commutation between servers to check computational progresses in multiple servers. DTA currently has communication between cloud servers and a mobile device. However, we extend the idea of data communication, so all the cloud servers can share their energy efficiency and estimated time to improve the performance of DTA.

LIST OF REFERENCES

[1] K. Lahiri, S. Dey, D. Panigrahi et al., "Battery-driven system design: a new frontier in low power design," in *Proceedings of the Asia and South Pacific Design Automation Conference*, pp. 261–267, Bangalore, India, 2002.

[2] R. Rao, S. Vrudhula, and D. N. Rakhmatov, "Battery modeling for energy-aware system design," *IEEE Computer*, vol. 36, no. 12, pp. 77–87, 2003.

[3] Karthik Kumar, Jibang Liu, Yung-Hsiang Lu, and Bharat Bhargava. 2013. A Survey of Computation Offloading for Mobile Systems. *Mob. Netw. Appl.* 18, 1 (February 2013), 129-140.

[4] FightAIDS@home. http://www.fightaidsathome.org. March, 2014

[5] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. SETI@home: an experiment in public-re- source computing. Communications of the ACM , 45(11):56– 61, 2002

[6] V. S. Pande, I. Baker, J. Chapman, S. Elmer, S. M. Larson, Y. M. Rhee, M. R. Shirts, C. D. Snow, E. J. Sorin, and B. Za- grovic. Atomistic protein folding simulations on the submil- lisecond time scale using worldwide distributed computing. *Peter Kollman Memorial Issue, Biopolymers* , 68(1):91–109, 2003

[7] Yu-Ju Hong; Kumar, K.; Yung-Hsiang Lu, "Energy efficient content-based image retrieval for mobile systems," *Circuits and Systems, 2009. ISCAS 2009. IEEE International Symposium on* , vol., no., pp.1673,1676, 24-27 May 2009

[8] Nimmagadda, Y.; Kumar, K.; Yung-Hsiang Lu; Lee, C. S G, "Real-time moving object recognition and tracking using computation offloading," *Intelligent Robots and*

Systems (IROS), 2010 IEEE/RSJ International Conference on , vol., no., pp.2449,2455, 18-22 Oct. 2010

[9] Wolski, R.; Gurun, S.; Krintz, C.; Nurmi, D., "Using bandwidth data to make computation offloading decisions," *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on* , vol., no., pp.1,8, 14-18 April 2008

[10] H. Dinh, C. Lee, D. Niyato,W. Ping, " A Survey of Mobile Cloud Computing: Architecture, Applications, and Approaches," *Wireless Communications and Mobile Computing* , Vol. 13, no. , pp. 1587–1611, 11 OCT 2011.

[11] R. Niu, W. Song, Y. Liu, "An Energy-Efficient Multisite Offloading Algorithm for Mobile Devices," *International Journal of Distributed Sensor Networks*, Vol. 2013, no. , 6, 20 February 2013.

[12] Meng Xu; Lizhen Cui; Haiyang Wang; Yanbing Bi, "A Multiple QoS Constrained Scheduling Strategy of Multiple Workflows for Cloud Computing," *Parallel and Distributed Processing with Applications, 2009 IEEE International Symposium on* , vol., no., pp.629,634, 10-12 Aug. 2009

[13] Rouse, Margaret. "Platform as a Service (PaaS)." *Platform as a Service (PaaS)*. Techtarget.com, 04 Aug. 2014.Web. 05 Feb. 2014. <http://searchcloudcomputing.techtarget.com/definition/Platform-as-a-Service-PaaS>.

[14] Sedgewick, Robert, and Kevin Wayne. "4.2 Sorting and Searching." Sorting and Searching. Cs.princeton.edu, 07 June 2012. Web. 03 Jan. 2014. http://introcs.cs.princeton.edu/java/42sort/Gaussian.java

[15] Pandey, S.; Linlin Wu; Guru, S.M.; Buyya, R., "A Particle Swarm Optimization-Based Heuristic for Scheduling Workflow Applications in Cloud Computing Environments," *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference* on , vol., no., pp.400,407, 20-23 April 2010